CSE474 Embedded Systems

Lab 4

Student ID	Name	Email
1471626	Daniel Sullivan	dtsull16@uw.edu
1826649	Jeffrey Lee	cn1207@uw.edu

1 Procedure

1.1 Task 1

Task 1 was fairly straight forward since it pseudocode was already laid out for the function in SSD2119.c. The only thing we needed to do was look up the values for the indicated registers in the data sheet. This got our LCD screen successfully running and displaying images with LCD_ColorFill and the LCD_Draw functions.

1.2 Task 2

Task 2 was also simple because we could reuse a lot of our code from Lab3. All the ADC initialization and reading was the same as Lab 3 Task 1. The only difference we made was modifying the print functions in SSD2119.c to print the resulting temperature readings.

1.3 Task 3

To begin Task 3, we first set up red, yellow, and green circles with rectangular screen buttons below them for the traffic light and traffic light controls. This was fairly simple using the LCD_Draw and LCD printf functions. The main challenge came from reading user input to respond to the buttons. We used a similar format as the Lab3 traffic light, with a 5 second timer interrupt switching between the states and a 2 second timer interrupt triggering if the user was still holding down the button for 2 seconds. The main difference is that instead of starting the 2 second timer on a button interrupt, we had to continuously check if the screen was being pressed and, upon being pressed, detect whether the user was pressing a button.

2 Results

2.1 Task 1

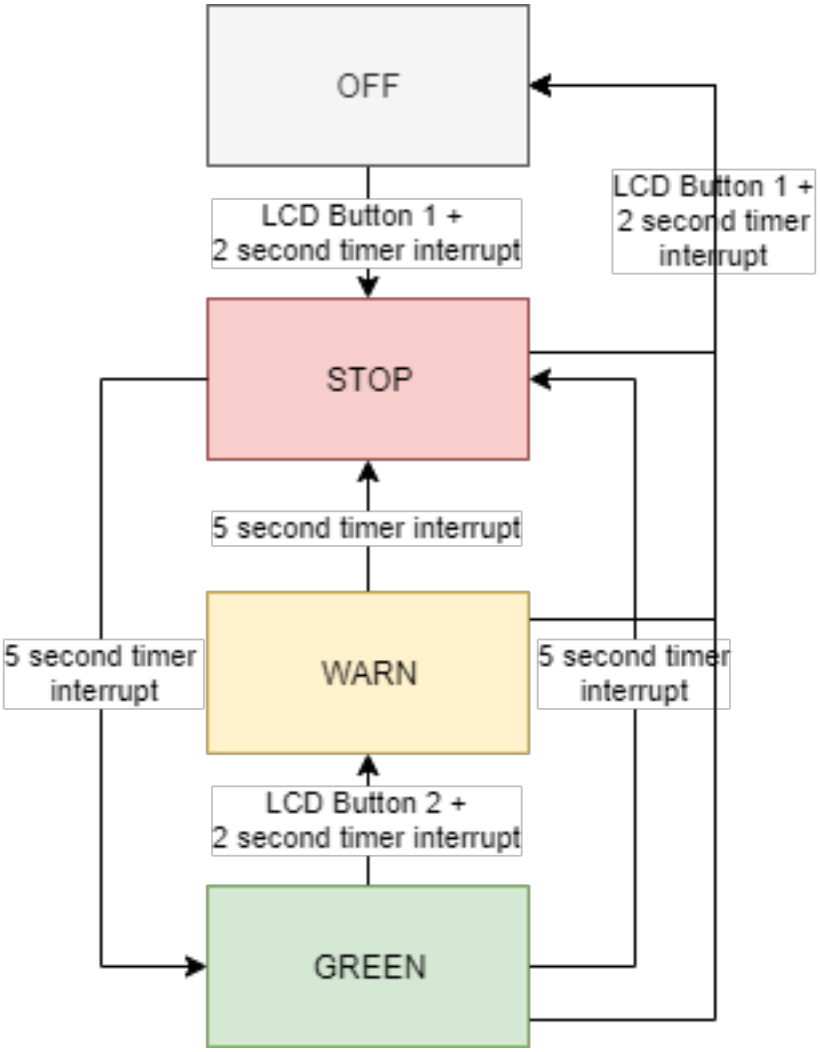
Our LCD_GPIOInit function successfully allowed us to set up the LCD and print to the screen, fulfilling Task 1 requirements. Since this was just an initialization function, it did not have any state transitions.

2.2 Task 2

Our Task 2 implementation successfully prints out continuous temperature readings from the board's internal thermometer and fulfills the requirements for Task 2. Since it is always in one continuous state of printing temperature readings, I did not include a state diagram.

2.3 Task 3

Task 3 successfully displays traffic light lights based on its state and presents 2 user buttons on the screen that the user can change the state with. The main difficulty comes from reading the position of user clicks on the screen. The LCD driver code provided for reading the position of screen touches is fairly convoluted and provides inconsistent readings so sometimes several attempts are needed to get a button to activate. Other than that difficulty, the program successfully switches between STOP and GO states every 5 seconds, will set the lights to WARN upon pressing the pedestrian button, and will only activate buttons upon holding them for 2 seconds. This fulfills all requirements for Task 3 represented in the following state diagram.



3 Problems Encountered and Feedback

3.1 Task 1

We encountered very few problems with Task 1 since we were already familiar with setting up ports from the data sheet and the psudocode was very straight forward.

3.2 Task 2

Similarly, we encountered few problems in Task 2 since we had already written code to read temperature with the ADC in Lab 3 and the LCD print and draw functions in the driver code were very easy to use.

3.3 Task 3

Task 3 presented a lot of challenges, mainly with reading user input from the LCD screen. The driver code for reading user input was very difficult to understand. It contained a lot of data transformations with no explanation, had several functions commented out or partially commented out, and had a misleading interrupt handler partially commented out. We eventually figured out that we could not reliably use the interrupt handler to read ADC information so we simply continuously checked if the LCD screen was being pressed. Even once that was successful, it was very difficult to detect where the user was pressing their finger. The readings would jump around drastically and had to correlation to the coordinates used to draw information on the screen. More documentation or clearer driver code would help immensely with handling user input.

4 Appendix

4.1 Lab 4

```
#include <stdint.h>
#define RED 0x2   // PF1
#define BLUE 0x4  // PF2
#define GREEN 0x8 // PF3
#define PORTA_RED 0x04 // PA2
#define PORTA_YELLOW 0x08 // PA 3
#define PORTA_GREEN 0x10 // PA4
#define SWITCHES 0x11 // PF0 and PF4
#define GPIO_PORTF 0x20
#define GPIO_PORTA 0x01


// system control register
// according to data sheet "legacy software" the offset is 0x108 instead of 0x60
// base is 0x400F E000
#define RCGCGPIO (*((volatile uint32_t *) 0x400FE108))

// offset 0x400, base 0x4002 5000, p663
#define GPIO_PORTF_DIR (*((volatile uint32_t *) 0x40025400))

// stands for digital enable page 682
// base 0x4002 5000
// offset 0x51C
#define GPIO_PORTF_DEN (*((volatile uint32_t *) 0x4002551C))

// the base for GPIODIR register is 0x40025000
// in order for GPIODATA to write, bit [9:2] must be set
// therefore 0x3FC = 0b0011 1111 1100 is used
#define GPIO_PORTF_DATA (*((volatile uint32_t *) 0x400253FC))


// GPIOPUR pull up resistor
// base 0x4002.5000, offset 0x510
#define GPIO_PORTF_PUR (*((volatile uint32_t *) 0x40025510))
```

```
// GPIOCR commit
// base 0x4002.5000, offset 0x524
#define GPIOCR (*((volatile uint32_t *) 0x40025524))


// GPIOLOCK
// base 0x4002.5000, offset 0x520
#define GPIO_LOCK (*(( volatile uint32_t *)0x40025520))

// GPIOPCTL port control register
// base 0x4000.4000, offset 0x52C
#define GPIO_PORTA_PCTL (*((volatile uint32_t *) 0x4000452C))

// GPIO analog mode select p.687
// base 0x4000.4000, offset 0x528
#define GPIO_PORTA_AMSEL (*((volatile uint32_t *) 0x40004528))

// GPIO PORTA direction p663
// base 0x4000.4000, offset 0x400
#define GPIO_PORTA_DIR (*((volatile uint32_t *) 0x40004400))

// GPIO alternate function select p.672
// base 0x4000.4000, offset 0x420
// if set the associated pin functions as a peripheral signal and is
// control by the alternate hardware function
#define GPIO_PORTA_AFSEL (*((volatile uint32_t *)0x40004420))

// GPIO digital enable p.682
// base 0x4000.4000, offset 0x51C
#define GPIO_PORTA_DEN (*((volatile uint32_t *)0x4000451C))

// GPIO PORTA DATA p.662
// base 0x4002.5000, offset
#define GPIO_PORTA_DATA (*((volatile uint32_t *)0x400043FC))

// base 0x400F.E000, offset 0x604
// p.338
#define RCGCTIMER (*((volatile uint32_t *) 0x400FE604))

// base 0x4003.0000, offset 0x00C
// p.737 timer0 control
#define TIMER0_CTL (*((volatile uint32_t *) 0x4003000C))
// base 0x4003.1000
#define TIMER1_CTL (*((volatile uint32_t *) 0x4003100C))

// base 0x4003.0000, offset 0x000
// p.727 GP timer config concatenated, individual, split
#define TIMTER0_CONFIG (*((volatile uint32_t *) 0x40030000))
// base 0x4003.1000
#define TIMTER1_CONFIG (*((volatile uint32_t *) 0x40031000))
```

```
// base 0x4003.0000, offset 0x004
// p.729 timer A mode
#define TIMER0_TAMR (*((volatile uint32_t *) 0x40030004))
// base 0x4003.1000
#define TIMER1_TAMR (*((volatile uint32_t *) 0x40031004))

// base 0x4003.0000, offset 0x028
// p.756 Timer A interval load
#define TIMER0_TAILR (*((volatile uint32_t *) 0x40030028))
// base 0x4003.1000
#define TIMER1_TAILR (*((volatile uint32_t *) 0x40031028))

// base 0x4003.0000, offset 0x01C
// p.748 raw interrupt status
#define TIMER0_RIS (*((volatile uint32_t *) 0x4003001C))
// base 0x4003.1000
#define TIMER1_RIS (*((volatile uint32_t *) 0x4003101C))

// base 0x4003.0000, offset 0x024
// p.754 interrupt clear
#define TIMER0_ICR (*((volatile uint32_t *) 0x40030024))
// base 0x4003.1000
#define TIMER1_ICR (*((volatile uint32_t *) 0x40031024))

// base 0x4003.0000, offset 0x018
// GPTM interrupt mask
#define TIMER0_IMR (*((volatile uint32_t *) 0x40030018))
// base 0x4003.1000, offset 0x018
#define TIMER1_IMR (*((volatile uint32_t *) 0x40031018))


// base 0xE000.E000, offset 0x100
// set enable
#define NVIC_EN0 (*((volatile uint32_t *) 0xE000E100))

// base 0x4002.5000, offset 0x418
// masked interrupt status
#define GPIO_PORTF_MIS (*((volatile uint32_t *) 0x40025418))
// base 0x4000.4000, offset 0x418
#define GPIO_PORTA_MIS (*((volatile uint32_t *) 0x40004418))


// base 0x4002.5000, offset 0x410
// interrupt mask
#define GPIO_PORTF_IM (*((volatile uint32_t *) 0x40025410))
// base 0x4000.4000, offset 0x410
#define GPIO_PORTA_IM (*((volatile uint32_t *) 0x40004410))

// base 0x4002.5000, offset 0x41C
// interrupt clear
#define GPIO_PORTF_ICR (*((volatile uint32_t *) 0x4002541C))
```

```
// base 0x4000.4000, offset 0x41C
#define GPIO_PORTA_ICR (*((volatile uint32_t *) 0x4000441C))

// base 0x4003.0000, offset 0x020
// timer interrupt status
#define TIMER0_MIS (*((volatile uint32_t *) 0x40030020))
// base 0x4003.1000
#define TIMER1_MIS (*((volatile uint32_t *) 0x40031020))

// base 0x400F.E000, offset 0x638
// adc control
#define RCGCADC (*((volatile uint32_t *) 0x400FE638))

// base 0x4003.8000, offset 0x000
// adc seqencer adcpssi ADCACTSS
#define ADC0_SEQ (*((volatile uint32_t *) 0x40038000))

// base 0x4003.8000, offset 0x014
// adc multiplexer adcemux
#define ADC0_MUX (*((volatile uint32_t *) 0x40038014))

// base  0x4003.8000, offset 0x0A0
// adc sample sequencer mux ADCSSMUX3
#define ADC0_SS_MUX3 (*((volatile uint32_t *) 0x400380A0))

// base  0x4003.8000, offset 0x0A4
// adc sample sequencer control ADCSSCTL3
#define ADC0_SS_CTL3 (*((volatile uint32_t *) 0x400380A4))


// base  0x4003.8000, offset 0x0A4
// adc sample sequencer control ADCSSCTL3
#define ADC0_SS_CTL3 (*((volatile uint32_t *) 0x400380A4))


// base  0x4003.8000, offset 0x028
// adc sample sequencer initiate ADCPSSI
#define ADC0_SS_INIT (*((volatile uint32_t *) 0x40038028))


// base  0x4003.8000, offset 0x004
// adc raw interrupt status ADCRIS
#define ADC0_RIS (*((volatile uint32_t *) 0x40038004))

// base  0x400F.E000, offset 0x070
// run-mode clock config
#define RCC2 (*((volatile uint32_t *) 0x400FE070))

// base  0x400F.E000, offset 0x060
// run-mode clock config
#define RCC (*((volatile uint32_t *) 0x400FE060))
```

```
// base  0x4003.8000, offset 0x00C
// interrupt status and clear ADCISC
#define ADC0_ISC (*((volatile uint32_t *) 0x4003800C))

// base  0x400F.E000, offset 0x050
// raw interrupt status
#define RIS (*((volatile uint32_t *) 0x400FE050))

// base  0x400F.E000, offset 0x058
// masked interrupt status and clear
#define MISC (*((volatile uint32_t *) 0x400FE058))

// base  0x4003.8000, offset 0x008
// adc interrupt mask
#define ADC0_IM (*((volatile uint32_t *) 0x40038008))

// base  0x4003.8000, offset 0x0A8
// sequencer result ADCSSFIFO3
#define ADC0_FIFO3 (*((volatile uint32_t *) 0x400380A8))

// base  0x400F.E000, offset 0x618
// UART clock gating
#define RCGCUART (*((volatile uint32_t *) 0x400FE618))

// base  0x4000.C000, offset 0x030
// UART control UARTCTL
#define UART0_CTL (*((volatile uint32_t *) 0x4000C030))

// base  0x4000.C000, offset 0x024
// UART int baud rate divisor UARTIBRD
#define UART0_IBRD (*((volatile uint32_t *) 0x4000C024))

// base  0x4000.C000, offset 0x028
// UART frac baud rate divisor UARTFBRD
#define UART0_FBRD (*((volatile uint32_t *) 0x4000C028))
// uart line control UARTLCRH
#define UART0_LCRH (*((volatile uint32_t *) 0x4000C02C))
// uart clock configuration UARTCC
#define UART0_CC (*((volatile uint32_t *) 0x4000CFC8))
// uart flag UARTFR
#define UART0_FR (*((volatile uint32_t *) 0x4000C018))
// uart data UARTDR
#define UART0_DR (*((volatile uint32_t *) 0x4000C000))


// UART register
#define UART_R0 0x1
#define UARTEN 0x1
#define WLEN_8BITS 0x30
#define FEN 0x10 // FIFO enable
```

```
#define STP2 0x40 // stop bit 2
#define PEN 0x20 // parity bit enable


// PLL register
#define RCC_BYPASS 0x800
#define RCC2_BYPASS2 0x800
#define RCC_USESYSDIV 0x400000
#define RCC_XTAL 0x7C0
#define RCC_XTAL_4MHZ 0x180
#define RCC_XTAL_16MHZ 0x540
#define RCC_OSCSRC 0x30
#define RCC_OSCSRC_MAIN 0x00
#define RCC2_OSCSRC2 0x70
#define RCC2_OSCSRC2_MAIN 0x00
#define RCC_PWRDN 0x2000
#define RCC2_PWRDN2 0x2000
#define RCC2_USERCC2 0x80000000
#define RCC2_DIV400 0x40000000

// adc registers
#define ASEN3 0x8
#define EMUX_EM3 0xF000
#define EMUX_TIMER 0x5000

// timer register
#define TAOTE 0x20
#define TAEN 0x1

#define Bus80MHz     4

// LCD UI define
#define SWITCH1_XPOS 55
#define SWITCH2_XPOS 185
#define SWITCH_YPOS 140
#define SWITCH_WIDTH 80
#define SWITCH_HEIGHT 60

#define DETECT_SWITCH1_XPOS_LEFT 300
#define DETECT_SWITCH1_XPOS_RIGHT 120
#define DETECT_SWITCH1_YPOS_UP 100
#define DETECT_SWITCH1_YPOS_DOWN 180

#define DETECT_SWITCH2_XPOS_LEFT 100
#define DETECT_SWITCH2_XPOS_RIGHT 40
#define DETECT_SWITCH2_YPOS_UP 135
#define DETECT_SWITCH2_YPOS_DOWN 185


#define  RED_XPOS 70
#define  YELLOW_XPOS 160
```

```
#define  GREEN_XPOS 250
#define RADIUS 40
#define  BULB_YPOS 80

// SSD2119.c

// Runs on LM4F120/TM4C123
// Driver for the SSD2119 interface on a Kentec 320x240x16 BoosterPack
// - Uses all 8 bits on PortB for writing data to LCD
//   and bits 4-7 on Port A for control signals
//
//
//  Data pin assignments:
//  PB0-7   LCD parallel data input
//
//  Control pin assignments:
//  PA4     RD  Read control signal             -------------------------
//  PA5     WR  Write control signal            | PA7 | PA6 | PA5 | PA4 |
//  PA6     RS  Register/Data select signal     | CS  | RS  | WR  | RD  |
//  PA7     CS  Chip select signal              -------------------------
//
//  Touchpad pin assignments:
//  PA2     Y-                                  -------------  -------------
//  PA3     X-                                  | PA3 | PA2 |  | PE5 | PE4 |
//  PE4     X+    AIN9                          | X-  | Y-  |  | Y+  | X+  |
//  PE5     Y+    AIN8                          -------------  -------------
//
//  Touchscreen resistance measurements:
//  -------------------
//  |1               2|                XN->YP       XP->YN
//  |                 |         1       1150         1400
//  |                 |         2       640           800
//  |        5        |         3       1400         1100
//  |                 |         4       870           580
//  |3               4|         5       1000          960
//  -------------------
//
//  XP->XN = 651

#include <stdint.h>
#include "tm4c123gh6pm.h"
//#include "inc/tm4c123gh6pm.h"
#include "SSD2119.h"
#include "my_header.h"
#include <stdbool.h>


// 4 bit Color   red,green,blue to 16 bit color
// bits 15-11 5 bit red
// bits 10-5  6-bit green
// bits  4-0  5-bit blue
unsigned short const Color4[16] = {
```

```
  0,                                                //0 â black                    (#
 ((0x00>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3),  //1 â blue                     (#
 ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0x00>>3),  //2 â green                    (#
 ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0xAA>>3),  //3 â cyan                     (#
 ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0x00>>3),  //4 â red                      (#
 ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3),  //5 â magenta                  (#
 ((0xAA>>3)<<11) | ((0x55>>2)<<5) | (0x00>>3),  //6 â brown                    (#
 ((0xAA>>3)<<11) | ((0xAA>>2)<<5) | (0xAA>>3),  //7 â white / light gray       (#
 ((0x55>>3)<<11) | ((0x55>>2)<<5) | (0x55>>3),  //8 â dark gray /bright black (#
 ((0x55>>3)<<11) | ((0x55>>2)<<5) | (0xFF>>3),  //9 â bright blue              (#
 ((0x55>>3)<<11) | ((0xFF>>2)<<5) | (0x55>>3),  //10 â bright green           (#
 ((0x55>>3)<<11) | ((0xFF>>2)<<5) | (0xFF>>3),  //11 â bright cyan            (#
 ((0xFF>>3)<<11) | ((0x55>>2)<<5) | (0x55>>3),  //12 â bright red             (#
 ((0xFF>>3)<<11) | ((0x55>>2)<<5) | (0xFF>>3),  //13 â bright magenta         (#
 ((0xFF>>3)<<11) | ((0xFF>>2)<<5) | (0x55>>3),  //14 â bright yellow          (#
 ((0xFF>>3)<<11) | ((0xFF>>2)<<5) | (0xFF>>3)   //15 â bright white           (#
};


unsigned short cursorX;
unsigned short cursorY;
unsigned short textColor;

typedef struct {
    short x;
    short y;
} coord;

// dimensions of the LCD in pixels
#define LCD_HEIGHT 240
#define LCD_WIDTH 320



// converts 24bit RGB color to display color
//#define CONVERT24BPP(c) ( (((c) & 0x00f80000) >> 8) | (((c) & 0x0000fc00) >> 5
#define CONVERT24BPP(c) ( (((c) & 0x00f80000) >> 8) | (((c) & 0x0000fc00) >> 5)

// converts 8bit greyscale to display color
#define CONVERT8BPP(c)  ( (((c) >> 3) << 11) | (((c) >> 2) << 5 ) | ((c) >> 3) )

// converts 4bit greyscale to display color
#define CONVERT4BPP(c)  ( ((c) << 12) | ((c) << 7 ) | ((c) << 1) )

// define BMP offsets
#define BMP_WIDTH_OFFSET        0x0012
#define BMP_HEIGHT_OFFSET       0x0016
#define BMP_DATA_OFFSET         0x000A
#define BMP_BPP_OFFSET          0x001C

// define command codes
```

```
#define SSD2119_DEVICE_CODE_READ_REG    0x00
#define SSD2119_OSC_START_REG           0x00
#define SSD2119_OUTPUT_CTRL_REG         0x01
#define SSD2119_LCD_DRIVE_AC_CTRL_REG   0x02
#define SSD2119_PWR_CTRL_1_REG          0x03
#define SSD2119_DISPLAY_CTRL_REG        0x07
#define SSD2119_FRAME_CYCLE_CTRL_REG    0x0B
#define SSD2119_PWR_CTRL_2_REG          0x0C
#define SSD2119_PWR_CTRL_3_REG          0x0D
#define SSD2119_PWR_CTRL_4_REG          0x0E
#define SSD2119_GATE_SCAN_START_REG     0x0F
#define SSD2119_SLEEP_MODE_REG          0x10
#define SSD2119_ENTRY_MODE_REG          0x11
#define SSD2119_GEN_IF_CTRL_REG         0x15
#define SSD2119_PWR_CTRL_5_REG          0x1E
#define SSD2119_RAM_DATA_REG            0x22
#define SSD2119_FRAME_FREQ_REG          0x25
#define SSD2119_VCOM_OTP_1_REG          0x28
#define SSD2119_VCOM_OTP_2_REG          0x29
#define SSD2119_GAMMA_CTRL_1_REG        0x30
#define SSD2119_GAMMA_CTRL_2_REG        0x31
#define SSD2119_GAMMA_CTRL_3_REG        0x32
#define SSD2119_GAMMA_CTRL_4_REG        0x33
#define SSD2119_GAMMA_CTRL_5_REG        0x34
#define SSD2119_GAMMA_CTRL_6_REG        0x35
#define SSD2119_GAMMA_CTRL_7_REG        0x36
#define SSD2119_GAMMA_CTRL_8_REG        0x37
#define SSD2119_GAMMA_CTRL_9_REG        0x3A
#define SSD2119_GAMMA_CTRL_10_REG       0x3B
#define SSD2119_V_RAM_POS_REG           0x44
#define SSD2119_H_RAM_START_REG         0x45
#define SSD2119_H_RAM_END_REG           0x46
#define SSD2119_X_RAM_ADDR_REG          0x4E
#define SSD2119_Y_RAM_ADDR_REG          0x4F
#define ENTRY_MODE_DEFAULT              0x6830


// number of 5x8 characters that will fit on the screen
#define MAX_CHARS_X     53
#define MAX_CHARS_Y     26

// entry mode macros
#define HORIZ_DIRECTION     0x28
#define VERT_DIRECTION      0x20
#define ENTRY_MODE_DEFAULT  0x6830   // 0110.1000.0011.0000
#define MAKE_ENTRY_MODE(x) ((ENTRY_MODE_DEFAULT & 0xFF00) | (x))

// bit-banded addresses for port stuff
#define LCD_RD_PIN       (*((volatile unsigned long *)0x40004040))     // PA4
#define LCD_WR_PIN       (*((volatile unsigned long *)0x40004080))     // PA5
#define LCD_RS_PIN       (*((volatile unsigned long *)0x40004100))     // PA6
```

```
#define LCD_CS_PIN       (*((volatile unsigned long *)0x40004200))     // PA7
#define LCD_CTRL         (*((volatile unsigned long *)0x400043C0))     // PA4-7
#define LCD_DATA         (*((volatile unsigned long *)0x400053FC))     // PB0-7
// ************** ADC_Init *********************************
// - Initializes the ADC to use a specficed channel on SS3
// *********************************************************
// Input: channel number
// Output: none
// *********************************************************
void ADC_Init(void);

// ************** ADC_Read *********************************
// - Takes a sample from the ADC
// *********************************************************
// Input: none
// Output: sampled value from the ADC
// *********************************************************
unsigned long ADC_Read(void);

// ************** ADC_SetChannel ***************************
// - Configures the ADC to use a specific channel
// *********************************************************
// Input: none
// Output: none
// *********************************************************
void ADC_SetChannel(unsigned char channelNum);

/* ************************************************************
TODO: Please fill the information based on the pseudocode
**************************************************************/

// ************** TODO: LCD_GPIOInit ***************************
// - Initializes Port B to be used as the data bus and
//   Port A 4-7 as controller signals
// *********************************************************
void LCD_GPIOInit(void){
    unsigned long wait = 0;

    // SYSCTL_RCGCGPIO_R |= (1 << 1);
    SYSCTL_RCGC2_R |= (1 << 1);            // activate port B
    wait++;                                // wait for port activation
    wait++;                                // wait for port activation
    GPIO_PORTB_DIR_R |= 0xFF;              // make PB0-7 outputs
    GPIO_PORTB_AFSEL_R &= ~0xFF;            // disable alternate functions
    GPIO_PORTB_DEN_R |= 0xFF;              // enable digital I/O on PB0-7

    // activate control pins
    // SYSCTL_RCGCGPIO_R |= (1 << 0);
    SYSCTL_RCGC2_R |= (1 << 0);           // activate port A
    wait++;                               // wait for port activation
    wait++;                               // wait for port activation
```

```
    GPIO_PORTA_DIR_R |= 0xF0;              // make PA4-7 outputs
    GPIO_PORTA_AFSEL_R &= ~0xF0;           // disable alternate functions
    GPIO_PORTA_DEN_R |= 0xF0;             // enable digital I/O on PA4-7

    for (wait = 0; wait < 500; wait++) {}
}

// ************** LCD_WriteCommand ************************
// - Writes a command to the LCD controller
// - RS low during command write
// ********************************************************
//  PA4     RD  Read control signal             -------------------------
//  PA5     WR  Write control signal            | PA7 | PA6 | PA5 | PA4 |
//  PA6     RS  Register/Data select signal     | CS  | RS  | WR  | RD  |
//  PA7     CS  Chip select signal              -------------------------
void LCD_WriteCommand(unsigned char data){volatile unsigned long delay;
    LCD_CTRL = 0x30;  // Set CS=0, RS=0, WR=1, RD=1
    LCD_DATA = 0x00;  // Write 0 as MSB of command data
    delay++;
    LCD_CTRL = 0x10;  // Set WR low
    delay++;
    LCD_CTRL = 0x30;  // Set WR high
    LCD_DATA = data;  // Write data as LSB of command data
    delay++;
    LCD_CTRL = 0x10;  // Set WR low
    delay++;
    LCD_CTRL = 0xF0;  // Set all high
}

// ************** LCD_WriteData ***************************
// - Writes data to the LCD controller
// - RS high during data write
// ********************************************************
//  PA4     RD  Read control signal             -------------------------
//  PA5     WR  Write control signal            | PA7 | PA6 | PA5 | PA4 |
//  PA6     RS  Register/Data select signal     | CS  | RS  | WR  | RD  |
//  PA7     CS  Chip select signal              -------------------------
void LCD_WriteData(unsigned short data){volatile unsigned long delay;
    LCD_CTRL = 0x70; // CS low
    LCD_DATA = (data >> 8); // Write MSB to LCD data bus
    delay++;
    LCD_CTRL = 0x50; // Set WR low
    delay++;
    LCD_CTRL = 0x70; // Set WR high
    LCD_DATA = data; // Write LSB to LCD data bus
    delay++;
    LCD_CTRL = 0x50; // Set WR low
    delay++;
    LCD_CTRL = 0xF0; // Set CS, WR high
}
```

```
// ************** LCD_Init *********************************
// - Initializes the LCD
// - Command sequence verbatim from original driver
// *********************************************************
void LCD_Init(void){
    unsigned long count = 0;

    LCD_GPIOInit();

    // Enter sleep mode (if we are not already there).
    LCD_WriteCommand(SSD2119_SLEEP_MODE_REG);
    LCD_WriteData(0x0001);

    // Set initial power parameters.
    LCD_WriteCommand(SSD2119_PWR_CTRL_5_REG);
    LCD_WriteData(0x00BA);
    LCD_WriteCommand(SSD2119_VCOM_OTP_1_REG);
    LCD_WriteData(0x0006);

    // Start the oscillator.
    LCD_WriteCommand(SSD2119_OSC_START_REG);
    LCD_WriteData(0x0001);

    // Set pixel format and basic display orientation (scanning direction).
    LCD_WriteCommand(SSD2119_OUTPUT_CTRL_REG);
    LCD_WriteData(0x72EF);                                  //0x72EF = 0,0 in to
    LCD_WriteCommand(SSD2119_LCD_DRIVE_AC_CTRL_REG);        //0x30EF = 0,0 in bo
    LCD_WriteData(0x0600);                                  //0x32EF = 0,0 in to

    // Exit sleep mode.
    LCD_WriteCommand(SSD2119_SLEEP_MODE_REG);
    LCD_WriteData(0x0000);

    // Delay 30mS
    for (count = 0; count < 200000; count++) {}

    // Configure pixel color format and MCU interface parameters.
    LCD_WriteCommand(SSD2119_ENTRY_MODE_REG);
    LCD_WriteData(ENTRY_MODE_DEFAULT);

    // Enable the display.
    LCD_WriteCommand(SSD2119_DISPLAY_CTRL_REG);
    LCD_WriteData(0x0033);

    // Set VCIX2 voltage to 6.1V.
    LCD_WriteCommand(SSD2119_PWR_CTRL_2_REG);
    LCD_WriteData(0x0005);

    // Configure gamma correction.
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_1_REG);
    LCD_WriteData(0x0000);
```

```
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_2_REG);
    LCD_WriteData(0x0400);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_3_REG);
    LCD_WriteData(0x0106);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_4_REG);
    LCD_WriteData(0x0700);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_5_REG);
    LCD_WriteData(0x0002);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_6_REG);
    LCD_WriteData(0x0702);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_7_REG);
    LCD_WriteData(0x0707);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_8_REG);
    LCD_WriteData(0x0203);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_9_REG);
    LCD_WriteData(0x1400);
    LCD_WriteCommand(SSD2119_GAMMA_CTRL_10_REG);
    LCD_WriteData(0x0F03);

    // Configure Vlcd63 and VCOMl.
    LCD_WriteCommand(SSD2119_PWR_CTRL_3_REG);
    LCD_WriteData(0x0007);
    LCD_WriteCommand(SSD2119_PWR_CTRL_4_REG);
    LCD_WriteData(0x3100);

    // Set the display size and ensure that the GRAM window is set to allow
    // access to the full display buffer.
    LCD_WriteCommand(SSD2119_V_RAM_POS_REG);
    LCD_WriteData((LCD_HEIGHT-1) << 8);
    LCD_WriteCommand(SSD2119_H_RAM_START_REG);
    LCD_WriteData(0x0000);
    LCD_WriteCommand(SSD2119_H_RAM_END_REG);
    LCD_WriteData(LCD_WIDTH-1);
    LCD_WriteCommand(SSD2119_X_RAM_ADDR_REG);
    LCD_WriteData(0x00);
    LCD_WriteCommand(SSD2119_Y_RAM_ADDR_REG);
    LCD_WriteData(0x00);

    // Clear the contents of the display buffer.
    LCD_WriteCommand(SSD2119_RAM_DATA_REG);
    for(count = 0; count < (320 * 240); count++)
    {
        LCD_WriteData(0x0000);
    }

    // Set text cursor to top left of screen
    LCD_SetCursor(0, 0);

    // Set default text color to white
    LCD_SetTextColor(255, 255, 255);
}
```

```
// ************** convertColor ****************************
// - Converts 8-8-8 RGB values into 5-6-5 RGB
// ********************************************************
unsigned short convertColor(unsigned char r, unsigned char g, unsigned char b){
    return ((r>>3)<<11) | ((g>>2)<<5) | (b>>3);
}

// ************** LCD_ColorFill ***************************
// - Fills the screen with the specified color
// ********************************************************
void LCD_ColorFill(unsigned short color){
    LCD_DrawFilledRect(0, 0, LCD_WIDTH, LCD_HEIGHT, color);
}

// ************** abs **************************************
// - Returns the absolute value of an integer
// - Used to help with circle drawing
// ********************************************************
int abs(int a){
    if (a < 0) return -a;
    else return a;
}

//////////////////////////////////////////////////////////////////////////////
//                                  PRINTING FUNCTIONS
//////////////////////////////////////////////////////////////////////////////

// ************** LCD_PrintChar ***************************
// - Prints a character to the screen
// ********************************************************
void LCD_PrintChar(unsigned char data){
    unsigned char i,j,tempData;

    // Return cursor to new line if requested
    if (data == '\n') {
        LCD_SetCursor(0, cursorY + 9);
    }

    // Don't print characters outside of this range
    if (data < 0x20 || data > 0x7e) return;

    // If character would print beyond LCD_WIDTH, go to new line
    if (cursorX + 5 >= LCD_WIDTH) {
        LCD_SetCursor(0, cursorY + 9);
    }

    // If character would print beyond LCD_HEIGHT, return to top of screen
    if (cursorY + 8 >= LCD_HEIGHT) {
        LCD_SetCursor(cursorX, 0);
    }
```

```
    // Print our character
    for(i=0; i<5; i=i+1){
        tempData = ASCII[data - 0x20][i];
        for (j=0; j<8; j=j+1){
            // This would print transparent letters
            // if (tempData & 0x01) {
            //     LCD_DrawPixel(cursorX + i, cursorY + j, textColor);
            // }

            // This will overwrite the entire character block (non-transparent)
            LCD_DrawPixel(cursorX + i, cursorY + j, (tempData & 0x01) * textColo

            // Shift to our next pixel
            tempData = tempData >> 1;
        }
    }

    // Set cursor to next location
    LCD_SetCursor(cursorX + 6, cursorY);
}

// ************** LCD_PrintString ************************
// - Prints a string to the screen
// *******************************************************
void LCD_PrintString(char data[]){
    unsigned short i = 0;

    // While data[i] is not a null terminator, print out characters
    while (data[i] != 0){
        LCD_PrintChar(data[i]);
        i += 1;
    }
}

// ************** LCD_SetCursor ***************************
// - Sets character printing cursor position
// ********************************************************
void LCD_SetCursor(unsigned short xPos, unsigned short yPos){
    // Set the X address of the display cursor.
    cursorX = xPos;
//    LCD_WriteCommand(SSD2119_X_RAM_ADDR_REG);
//    LCD_WriteData(xPos);

    // Set the Y address of the display cursor.
    cursorY = yPos;
//    LCD_WriteCommand(SSD2119_Y_RAM_ADDR_REG);
//    LCD_WriteData(yPos);

}
```

```
// ************** LCD_Goto *********************************
// - Sets character printing cursor position in terms of
//   character positions rather than pixels.
// - Ignores invalid position requests.
// *********************************************************
void LCD_Goto(unsigned char x, unsigned char y){
    if (x > MAX_CHARS_X - 1 || y > MAX_CHARS_Y - 1) return;
    LCD_SetCursor(x * 6, y * 9);
}

// ************** LCD_SetTextColor ************************
// - Sets the color that characters will be printed in
// *********************************************************
void LCD_SetTextColor(unsigned char r, unsigned char g, unsigned char b){
    textColor = convertColor(r, g, b);
}

// ************** printf **********************************
// - Basic printf() implementation
// - Adapted from Craig Chase, EE312 printf() case study
// - Supports:
//   - %d   Signed decimal integer
//   - %c   Character
//   - %s   String of characters
//   - %f   Decimal floating point          (NYI)
//   - %x   Unsigned hexadecimal integer
//   - %b   Binary integer
//   - %%   A single % output
// *********************************************************
void printf(char fmt[], ...) {
unsigned char k = 0;
void* next_arg = &fmt + 1;
while (fmt[k] != 0) {
if (fmt[k] == '%') {                    // Special escape, look for next arg
if (fmt[k+1] == 'd') {              // Display integer
long* p = (long*) next_arg;
long x = *p;
next_arg = p + 1;
LCD_PrintInteger(x);
} else if (fmt[k+1] == 'c') {       // Display character
long* p = (long*) next_arg;
char x = *p;
next_arg = p + 1;
LCD_PrintChar(x);
} else if (fmt[k+1] == 's') {       // Display string
char** p = (char**) next_arg;
char* x = *p;
next_arg = p + 1;
LCD_PrintString(x);
  } else if (fmt[k+1] == 'f') {       // Display float (not yet working)
  float* p = (float*) next_arg;
```

```
  float x = *p;
  next_arg = p + 1;
  LCD_PrintFloat(x);
} else if (fmt[k+1] == 'x') {       // Display hexadecimal
long* p = (long*) next_arg;
long x = *p;
next_arg = p + 1;
LCD_PrintHex(x);
} else if (fmt[k+1] == 'b') {       // Display binary
long* p = (long*) next_arg;
long x = *p;
next_arg = p + 1;
LCD_PrintBinary(x);
} else if (fmt[k+1] == '%') {       // Display '%'
LCD_PrintChar('%');
} else {
// Otherwise, just ignore the unrecognized escape
}
k = k + 2;
} else {                                  // Normal output, just print the charact
LCD_PrintChar(fmt[k]);
k = k + 1;
}
}
}

// ************** LCD_PrintInteger ************************
// - Prints a signed integer to the screen
// *******************************************************
void LCD_PrintInteger(long n){
    unsigned char i = 0;
    unsigned char sign = ' ';
    unsigned char tempString[16];

    // If our number is 0, print 0
    if (n == 0) {
        LCD_PrintChar('0');
        return;
    }

    // If our number is negative, remember and unsign it
    if(n < 0){
        n = -n;
        sign = '-';
    }

    // Build our number string via repeated division
    while (n > 0){
        tempString[i] = (n % 10) + 48;
        n = n / 10;
        i += 1;
```

```
    }

    // Apply our sign if necessary
    if (sign == '-'){
        LCD_PrintChar('-');
    }

    // Print out our string in reverse order
    while (i){
        LCD_PrintChar(tempString[i-1]);
        i -= 1;
    }
}

// ************** LCD_PrintHex ****************************
// - Prints a number in hexidecimal format
// ********************************************************
void LCD_PrintHex(unsigned long n){
    unsigned char i = 0;
    unsigned char tempString[16];

    // Print hex prefix
    LCD_PrintString("0x");

    // If our number is 0, print 0
    if (n == 0) {
        LCD_PrintString("00");
        return;
    }

    // Build hexidecimal string via repeated division
    while (n > 0){
        tempString[i] = (n % 16) + 48;
        if (tempString[i] > 57) tempString[i] += 7;
        n = n / 16;
        i += 1;
    }

    // Print an even number of zeros
    if (i & 0x01) {
        tempString[i] = '0';
        i += 1;
    }

    // Print out our string in reverse order
    while (i){
        LCD_PrintChar(tempString[i-1]);
        i -= 1;
    }
}
```

```
// ************** LCD_PrintBinary ************************
// - Prints a number in binary format
// *******************************************************
void LCD_PrintBinary(unsigned long n){
    unsigned char i = 0;
    unsigned char j = 0;
    unsigned char tempString[32];

    // Print binary prefix
    LCD_PrintString("0b");

    // If our number is 0, print 0
    if (n == 0) {
        LCD_PrintString("0000");
        return;
    }

    // Build hexidecimal string via repeated division
    while (n > 0){
        tempString[i] = (n % 2) + 48;
        n = n / 2;
        i += 1;
    }

    // Print in nibbles
    for (j = 0; j < (i % 4); j++){
        tempString[i] = '0';
        i += 1;
    }

    // Print out our string in reverse order
    while (i){
        LCD_PrintChar(tempString[i-1]);
        i -= 1;
        // add nibble seperators
        if (i % 4 == 0 && i != 0) LCD_PrintChar('.');
    }
}

// ************** LCD_PrintFloat ************************
// - Prints a floating point number (doesn't work right now)
// *******************************************************
void LCD_PrintFloat(float num){
    long temp;

    // Decode exponent
     // printf ("binary = %b\n", num);
     // printf ("hex    = %x\n", num);

    temp = ((long)num);
//     printf ("exponent = %d\n", temp);
```

```
//    printf("%f\n", temp);
    temp = (long)(num * (1 << 12));
    printf("temperature (Celsius): %d.%d", temp >> 12,  (temp & 0xFFF) * 1000 /
    LCD_PrintChar('\n');

}

////////////////////////////////////////////////////////////////////////////////
//                                    DRAWING FUNCTIONS
////////////////////////////////////////////////////////////////////////////////

// ************** LCD_DrawPixel ***************************
// - Draws a 16-bit pixel on the screen
// ********************************************************
void LCD_DrawPixel(unsigned short x, unsigned short y, unsigned short color)
{
    // Set the X address of the display cursor.
    LCD_WriteCommand(SSD2119_X_RAM_ADDR_REG);
    LCD_WriteData(x);

    // Set the Y address of the display cursor.
    LCD_WriteCommand(SSD2119_Y_RAM_ADDR_REG);
    LCD_WriteData(y);

    // Write the pixel value.
    LCD_WriteCommand(SSD2119_RAM_DATA_REG);
    LCD_WriteData(color);
}

// ************** LCD_DrawPixelRGB ************************
// - Draws a 16-bit representation of a 24-bit color pixel
// ********************************************************
void LCD_DrawPixelRGB(unsigned short x, unsigned short y, unsigned char r, unsig
    LCD_DrawPixel(x, y, convertColor(r, g, b));
}

// ************** LCD_DrawLine ****************************
// - Draws a line using the Bresenham line algrorithm from
//   http://rosettacode.org/wiki/Bitmap/Bresenham%27s_line_algorithm
// ********************************************************
void LCD_DrawLine(unsigned short startX, unsigned short startY, unsigned short e
    short x0 = startX;
    short x1 = endX;
    short y0 = startY;
    short y1 = endY;
    // compute the sign and diff of x and y
    short dx = abs(x1-x0), sx = x0 < x1 ? 1 : -1;
    short dy = abs(y1-y0), sy = y0 < y1 ? 1 : -1;
    short err = (dx > dy ? dx : -dy) / 2, e2;

    for(;;){
```

```
        LCD_DrawPixel(x0, y0, color);
        if (x0==x1 && y0==y1) break;
        e2 = err;
        if (e2 >-dx) { err -= dy; x0 += sx; }
        if (e2 < dy) { err += dx; y0 += sy; }
    }
}

// ************** LCD_DrawRect ****************************
// - Draws a rectangle, top left corner at (x,y)
// ********************************************************
void LCD_DrawRect(unsigned short x, unsigned short y, short width, short height,
    LCD_DrawLine(x, y, x + width, y, color);
    LCD_DrawLine(x, y + 1, x, y + height - 1, color);
    LCD_DrawLine(x, y + height, x + width, y + height, color);
    LCD_DrawLine(x + width, y + 1, x + width, y + height - 1, color);
}

// ************** LCD_DrawFilledRect **********************
// - Draws a filled rectangle, top left corner at (x,y)
// ********************************************************
void LCD_DrawFilledRect(unsigned short x, unsigned short y, short width, short h
    int i, j;

    for (i = 0; i < height; i++) {
        // Set the X address of the display cursor.
        LCD_WriteCommand(SSD2119_X_RAM_ADDR_REG);
        LCD_WriteData(x);

        // Set the Y address of the display cursor.
        LCD_WriteCommand(SSD2119_Y_RAM_ADDR_REG);
        LCD_WriteData(y + i);

        LCD_WriteCommand(SSD2119_RAM_DATA_REG);
        for (j = 0; j < width; j++) {
            LCD_WriteData(color);
        }
    }
}

// ************** LCD_DrawCircle ************************
// - Draws a circle centered at (x0, y0)
// ********************************************************
void LCD_DrawCircle(unsigned short x0, unsigned short y0, unsigned short radius,
  int f = 1 - radius;
  int ddF_x = 1;
  int ddF_y = -2 * radius;
  int x = 0;
  int y = radius;

  LCD_DrawPixel(x0, y0 + radius, color);
```

```
  LCD_DrawPixel(x0, y0 - radius, color);
  LCD_DrawPixel(x0 + radius, y0, color);
  LCD_DrawPixel(x0 - radius, y0, color);

  while(x < y)
  {
    ddF_x = 2 * x + 1;
    ddF_y = -2 * y;
    f = x*x + y*y - radius*radius + 2*x - y + 1;
    if(f >= 0)
    {
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x;
    LCD_DrawPixel(x0 + x, y0 + y, color);
    LCD_DrawPixel(x0 - x, y0 + y, color);
    LCD_DrawPixel(x0 + x, y0 - y, color);
    LCD_DrawPixel(x0 - x, y0 - y, color);
    LCD_DrawPixel(x0 + y, y0 + x, color);
    LCD_DrawPixel(x0 - y, y0 + x, color);
    LCD_DrawPixel(x0 + y, y0 - x, color);
    LCD_DrawPixel(x0 - y, y0 - x, color);
  }
}

// ************** LCD_DrawFilledCircle ********************
// - Draws a filled circle centered at (x0, y0)
// ********************************************************
void LCD_DrawFilledCircle(unsigned short x0, unsigned short y0, unsigned short r
    short x = radius, y = 0;
    short radiusError = 1-x;
    short i = 0;

    while(x >= y)
    {

        //LCD_DrawLine(x0 + x, y0 + y, x0 - x, y0 + y, color);
        for (i = x0 - x; i < x0 + x; i++){
            LCD_DrawPixel(i, y0 + y, color);
        }

        //LCD_DrawLine(x0 + x, y0 - y, x0 - x, y0 - y, color);
        for (i = x0 - x; i < x0 + x; i++){
            LCD_DrawPixel(i, y0 - y, color);
        }

        //LCD_DrawLine(x0 + y, y0 + x, x0 + y, y0 - x, color);
```

```
        for (i = y0 - x; i < y0 + x; i++){
            LCD_DrawPixel(x0 + y, i, color);
        }

        //LCD_DrawLine(x0 - y, y0 + x, x0 - y, y0 - x, color);
        for (i = y0 - x; i < y0 + x; i++){
            LCD_DrawPixel(x0 - y, i, color);
        }

        y++;

        // Calculate whether we need to move inward a pixel
        if(radiusError<0) {
            radiusError += 2*y+1;
        } else {
            x--;
            radiusError += 2*(y-x+1);
        }
    }
}

// ************** LCD_DrawImage ***************************
// - Draws an image from memory
// - Image format is a plain byte array (no metadata)
// - User must specify:
//   - pointer to image data
//   - x, y location to draw image
//   - width and height of image
//   - bpp (bits per pixel) of image
//     - currently supports 4 and 8 bpp image data
// ********************************************************
void LCD_DrawImage(const unsigned char imgPtr[], unsigned short x, unsigned shor
    short i, j, pixelCount;

    pixelCount = 0;

    for (i = 0; i < height; i++) {
        // Set the X address of the display cursor.
        LCD_WriteCommand(SSD2119_X_RAM_ADDR_REG);
        LCD_WriteData(x);

        // Set the Y address of the display cursor.
        LCD_WriteCommand(SSD2119_Y_RAM_ADDR_REG);
        LCD_WriteData(y + i);

        LCD_WriteCommand(SSD2119_RAM_DATA_REG);

        switch (bpp){
            case 4:
            {
                for (j = 0; j < width/2; j++) {
```

```
                    unsigned char pixelData = imgPtr[pixelCount];
                    LCD_WriteData(CONVERT4BPP((pixelData&0xF0)>>4));
                    LCD_WriteData(CONVERT4BPP(pixelData&0x0F));
                    pixelCount++;
                }
            } break;
            case 8:
            {
                for (j = 0; j < width; j++) {
                    char pixelData = *imgPtr + (i*j) + j;
                    LCD_WriteData( CONVERT8BPP(j) );
                    LCD_WriteData( CONVERT8BPP(pixelData&0x0F) );
                }
            }
        };
    }
}

// ************** LCD_DrawBMP ****************************
// - Draws an image from memory
// - Image format is a BMP image stored in a byte array
// - Function attempts to resolve the following metadata
//   from the BMP format
//   - width
//   - height
//   - bpp
//   - location of image data within bmp data
// - User must specify:
//   - pointer to image data
//   - x, y location to draw image
// *******************************************************
void LCD_DrawBMP(const unsigned char* imgPtr, unsigned short x, unsigned short y
    short i, j, bpp;
    long width, height, dataOffset;
    const unsigned char* pixelOffset;

    // read BMP metadata
    width = *(imgPtr + BMP_WIDTH_OFFSET);
    height = *(imgPtr + BMP_HEIGHT_OFFSET);
    bpp = *(imgPtr + BMP_BPP_OFFSET);
    dataOffset = *(imgPtr + BMP_DATA_OFFSET);


    // debug info
//    printf("height: %d, width: %d, bpp %d", height, width, bpp);

    // setup pixel pointer
    pixelOffset = imgPtr + dataOffset;

    for (i = 0; i < height; i++) {
        // Set the X address of the display cursor.
```

```
        LCD_WriteCommand(SSD2119_X_RAM_ADDR_REG);
        LCD_WriteData(x);

        // Set the Y address of the display cursor.
        LCD_WriteCommand(SSD2119_Y_RAM_ADDR_REG);
        LCD_WriteData(y + height - i);

        LCD_WriteCommand(SSD2119_RAM_DATA_REG);

        switch(bpp){
            case 1:
            {   // unknown if working yet
                for (j = 0; j < width/8; j++) {
                    unsigned char pixelData = *(pixelOffset);
                    LCD_WriteData((pixelData&0x80)*0xFFFF);
                    LCD_WriteData((pixelData&0x40)*0xFFFF);
                    LCD_WriteData((pixelData&0x20)*0xFFFF);
                    LCD_WriteData((pixelData&0x10)*0xFFFF);
                    LCD_WriteData((pixelData&0x08)*0xFFFF);
                    LCD_WriteData((pixelData&0x04)*0xFFFF);
                    LCD_WriteData((pixelData&0x02)*0xFFFF);
                    LCD_WriteData((pixelData&0x01)*0xFFFF);
                    pixelOffset++;
                }break;
            }
            case 4:
            {   // working?
                for (j = 0; j < width/2; j++) {
                    unsigned char pixelData = *(pixelOffset);
//                    LCD_WriteData( CONVERT4BPP((pixelData&0xF0)>>4) );
//                    LCD_WriteData( CONVERT4BPP(pixelData&0x0F) );
                    LCD_WriteData( Color4[(pixelData&0xF0)>>4] );
                    LCD_WriteData( Color4[pixelData&0x0F] );
                    pixelOffset++;
                } break;
            }
            case 24:
            {   // seems to work
                for (j = 0; j < width; j++) {
                    // read 24bit RGB value into pixelData
                    unsigned long pixelData = *(pixelOffset) | *(pixelOffset + 1

                    // write RGB value to screen (passed through conversion macr
                    LCD_WriteData( CONVERT24BPP(pixelData) );

                    // increment pixel data pointer to next 24bit value
                    pixelOffset += 3;
                }
            }
        }
    }
```

```
}

#define TOUCH_YN       (*((volatile unsigned long *)0x40004010))     // PA2
#define TOUCH_XP       (*((volatile unsigned long *)0x40004020))     // PA3
#define TOUCH_XN       (*((volatile unsigned long *)0x40024040))     // PE4 / AI
#define TOUCH_YP       (*((volatile unsigned long *)0x40024080))     // PE5 / AI

#define PA2     0x04
#define PA3     0x08
#define PE4     0x10
#define PE5     0x20

#define NUM_SAMPLES 4
#define NUM_VALS_TO_AVG 8

#define XVAL_MIN    100
#define YVAL_MIN    150

unsigned char Touch_WaitForInput = 0;
short Touch_XVal;
short Touch_YVal;


void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts

//short xVals[NUM_VALS_TO_AVG];
//short yVals[NUM_VALS_TO_AVG];
//unsigned char filterCounter = 0;
//unsigned char numVals = 0;


//         YP / PE5 / AIN8
//      ---------------------
//      |                   |
//   XP |                   | XN
//  PA3 |                   | PE4
//      |                   | AIN9
//      |                   |
//      |                   |
//      ---------------------
//             YN / PA2




// **************  Touch_Init *********************************
// - Initializes the GPIO used for the touchpad
// ******************************************************
// Input: none
// Output: none
```

```
// ********************************************************
void Touch_Init(void){
    unsigned long wait = 0;
    // Initialize ADC for use with touchscreen
    ADC_Init();

    // Activate PORTA GPIO clock
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA;
    wait++;
    wait++;

//    // Configure PA2/PA3 for GPIO digital output
    GPIO_PORTA_DIR_R    |=  0x0C;
//   GPIO_PORTA_AFSEL_R  &= ~0x0C;
//   GPIO_PORTA_DEN_R    |=  0x0C;
//
   // Set XN and YN low
//   TOUCH_XN = 0x00;
//   TOUCH_YN = 0x00;
//
    // Activate PORTE GPIO clock
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOE;
    wait++;
    wait++;

//    // Configure PE4/PE5 for GPIO digital output
    GPIO_PORTE_DIR_R    |=  0x30;
//   GPIO_PORTE_AFSEL_R  &= ~0x30;
//   GPIO_PORTE_DEN_R    |=  0x30;
//   GPIO_PORTE_AMSEL_R  &= ~0x30;
//
   // Set XP and YP high
//   TOUCH_XP = 0xFF;
//   TOUCH_YP = 0xFF;
}

// ************** ADC_Init *********************************
// - Initializes the ADC to use a specficed channel on SS3
// - This one is different from section A, please write your own
//    part A code. We consider it is
// ********************************************************
// Input: channel number
// Output: none
// ********************************************************
void ADC_Init(void){
    long wait = 0;

    // Set bit 0 in SYSCTL_RCGCADC_R to enable ADC0
    SYSCTL_RCGCADC_R    |=  0x01;
    for (wait = 0; wait < 50; wait++){}
```

```
    SYSCTL_RCGCGPIO_R |= 0x10;

    // Set ADC sample to 125KS/s
    ADC0_PC_R = 0x01;

    // Disable all sequencers for configuration
    ADC0_ACTSS_R &= ~0x000F;

    // Set ADC0 SS3 to highest priority
    ADC0_SSPRI_R = 0x0123;

    // Set bits 12-15 to 0x00 to enable software trigger on SS3
    ADC0_EMUX_R &= ~0xF000;

    // Set sample channel for sequencer 3
    ADC0_SSMUX3_R &= 0xFFF0;
    ADC0_SSMUX3_R += 9;

    // TS0 = 0, IE0 = 1, END0 = 1, D0 = 0
    ADC0_SSCTL3_R = 0x006;

    // Disable ADC interrupts on SS3 by clearing bit 3
    ADC0_IM_R &= ~0x0008;

    // Re-enable sample sequencer 3
    ADC0_ACTSS_R |= 0x0008;
}

// ************** ADC_Read *********************************
// - Takes a sample from the ADC
// *********************************************************
// Input: none
// Output: sampled value from the ADC
// *********************************************************
unsigned long ADC_Read(void){
    unsigned long result;

    // Set bit 3 to trigger sample start
    ADC0_PSSI_R = 0x008;

    // Wait for SS3 RIS bit to be set to 1
    while((ADC0_RIS_R&0x08)==0){};

    // Read 12-bit result from ADC from FIFO
    result = ADC0_SSFIFO3_R&0xFFF;

    // Clear SS3 RIS bit to 0 to acknowledge completion
    ADC0_ISC_R = 0x0008;

    return result;
}
```

```
// ************** ADC_SetChannel ****************************
// - Configures the ADC to use a specific channel
// ********************************************************
// Input: none
// Output: none
// ********************************************************
void ADC_SetChannel(unsigned char channelNum){
    // Disable all sequencers for configuration
    ADC0_ACTSS_R &= ~0x000F;

    // Set sample channel for sequencer 3
    ADC0_SSMUX3_R &= 0xFFF0;
    ADC0_SSMUX3_R += channelNum;

    // Re-enable sample sequencer 3
    ADC0_ACTSS_R |= 0x0008;
}

// ************** ADC_ReadXVal ******************************
// -
// ********************************************************
// Input: none
// Output: none
// ********************************************************
unsigned long Touch_ReadX(void){
    long i = 0;
    long sum = 0;
    long result = 0;

    GPIO_PORTA_DATA_R   &= ~PA2;

    // Configure PA3 (XP) for GPIO HIGH
    GPIO_PORTA_AFSEL_R  &= ~PA3;       // ASFEL = 0
    GPIO_PORTA_DEN_R    |=  PA3;       // DEN = 1
    GPIO_PORTA_DIR_R    |=  PA3;       // DIR = 1
    GPIO_PORTA_DATA_R   |=  PA3;       // DATA = 1

    // Configure PE4 (XN) for GPIO LOW
    GPIO_PORTE_AFSEL_R  &= ~PE4;       // ASFEL = 0
    GPIO_PORTE_DEN_R    |=  PE4;       // DEN = 1
    GPIO_PORTE_DIR_R    |=  PE4;       // DIR = 1
    GPIO_PORTE_DATA_R   &= ~PE4;       // DATA = 0

    // Configure PA2 (YN) for analog hi-Z
    GPIO_PORTA_AFSEL_R  &= ~PA2;       // AFSEL = 0
    GPIO_PORTA_DEN_R    &= ~PA2;       // DEN = 0
    GPIO_PORTA_DIR_R    &= ~PA2;       // DIR = 0

    // Configure PE5 (YP) for ADC use
    GPIO_PORTE_AFSEL_R  |=  PE5;       // AFSEL = 1
```

```
    GPIO_PORTE_DEN_R    &= ~PE5;        // DEN = 0
    GPIO_PORTE_AMSEL_R  |=  PE5;        // AMSEL = 1

    // Configure ADC to read from AIN8 (PE5, YP)
    ADC_SetChannel(8);

    // Take some samples to average
    for (i = 0; i < NUM_SAMPLES; i++){
        sum += ADC_Read();
    }

    GPIO_PORTE_AMSEL_R  &= ~PE5;        // AMSEL = 0

    // Compute average
    result = sum / NUM_SAMPLES;

    Touch_XVal = result;

    return result;
}

// ************** ADC_ReadYVal *****************************
// -
// ********************************************************
// Input: none
// Output: none
// ********************************************************
unsigned long Touch_ReadY(void){
    long i = 0;
    long sum = 0;
    long result = 0;

    // Configure PE5 (YP) for GPIO HIGH
    GPIO_PORTE_AFSEL_R  &= ~PE5;        // ASFEL = 0
    GPIO_PORTE_DEN_R    |=  PE5;        // DEN = 1
    GPIO_PORTE_DIR_R    |=  PE5;        // DIR = 1
    GPIO_PORTE_DATA_R   |=  PE5;        // DATA = 1
//    TOUCH_XP = 0xFF;

    // Configure PA2 (YN) for GPIO LOW
    GPIO_PORTA_AFSEL_R  &= ~PA2;        // ASFEL = 0
    GPIO_PORTA_DEN_R    |=  PA2;        // DEN = 1
    GPIO_PORTA_DIR_R    |=  PA2;        // DIR = 1
    GPIO_PORTA_DATA_R   &= ~PA2;        // DATA = 0

    // Configure PA3 (XP) for analog hi-Z
    GPIO_PORTA_AFSEL_R  &= ~PA3;        // AFSEL = 0
    GPIO_PORTA_DEN_R    &= ~PA3;        // DEN = 0
    GPIO_PORTA_DIR_R    &= ~PA3;        // DIR = 0

    // Configure PE4 (XN) for ADC use
```

```
    GPIO_PORTE_AFSEL_R  |=  PE4;       // AFSEL = 1
    GPIO_PORTE_DEN_R    &= ~PE4;       // DEN = 0
    GPIO_PORTE_AMSEL_R  |=  PE4;       // AMSEL = 1

    // Configure ADC to read from AIN9 (PE4, XN)
    ADC_SetChannel(9);

    // Take some samples to average
    for (i = 0; i < NUM_SAMPLES; i++){
        sum += ADC_Read();
    }

    // Compute average
    result = sum / NUM_SAMPLES;

    Touch_YVal = result;

    return result;
}

// ************** ADC_ReadZ1 ********************************
// -
// ***********************************************************
// Input: none
// Output: none
// ***********************************************************
unsigned long Touch_ReadZ1(void){
    long i = 0;
    long sum = 0;
    long result = 0;

    // Configure PA2 (YN) for GPIO HIGH
    GPIO_PORTA_AFSEL_R  &= ~PA2;       // ASFEL = 0
    GPIO_PORTA_DEN_R    |=  PA2;       // DEN = 1
    GPIO_PORTA_DIR_R    |=  PA2;       // DIR = 1
    GPIO_PORTA_DATA_R   |=  PA2;       // DATA = 1

    // Configure PA3 (XP) for GPIO LOW
    GPIO_PORTA_AFSEL_R  &= ~PA3;       // ASFEL = 0
    GPIO_PORTA_DEN_R    |=  PA3;       // DEN = 1
    GPIO_PORTA_DIR_R    |=  PA3;       // DIR = 1
    GPIO_PORTA_DATA_R   &= ~PA3;       // DATA = 0

    // Configure PE5 (YP) for analog hi-Z
    GPIO_PORTE_AFSEL_R  &= ~PE5;       // AFSEL = 0
    GPIO_PORTE_DEN_R    &= ~PE5;       // DEN = 0
    GPIO_PORTE_DIR_R    &= ~PE5;       // DIR = 0

    // Configure PE4 (XN) for ADC use
    GPIO_PORTE_AMSEL_R  |=  PE4;       // AMSEL = 1
    GPIO_PORTE_AFSEL_R  |=  PE4;       // AFSEL = 1
```

```
    GPIO_PORTE_DEN_R    &= ~PE4;       // DEN = 0
    GPIO_PORTE_DIR_R    &= ~PE4;       // DIR = 0

    // Configure ADC to read from AIN9 (PE4, XN)
    ADC_SetChannel(9);

    // Take some samples to average
    for (i = 0; i < NUM_SAMPLES; i++){
        sum += ADC_Read();
    }

    // Compute average
    result = sum / NUM_SAMPLES;

    return result;
}

// ************** ADC_ReadZ1 ********************************
// -
// ****************************************************
// Input: none
// Output: none
// ****************************************************
unsigned long Touch_ReadZ2(void){
    long i = 0;
    long sum = 0;
    long result = 0;

    // Configure PA2 (YN) for GPIO HIGH
    GPIO_PORTA_AFSEL_R  &= ~PA2;       // ASFEL = 0
    GPIO_PORTA_DEN_R    |=  PA2;       // DEN = 1
    GPIO_PORTA_DIR_R    |=  PA2;       // DIR = 1
    GPIO_PORTA_DATA_R   |=  PA2;       // DATA = 1

    // Configure PA3 (XP) for GPIO LOW
    GPIO_PORTA_AFSEL_R  &= ~PA3;       // ASFEL = 0
    GPIO_PORTA_DEN_R    |=  PA3;       // DEN = 1
    GPIO_PORTA_DIR_R    |=  PA3;       // DIR = 1
    GPIO_PORTA_DATA_R   &= ~PA3;       // DATA = 0

    // Configure PE4 (XN) for analog hi-Z
    GPIO_PORTE_AFSEL_R  &= ~PE4;       // AFSEL = 0
    GPIO_PORTE_DEN_R    &= ~PE4;       // DEN = 0
    GPIO_PORTE_DIR_R    &= ~PE4;       // DIR = 0
    GPIO_PORTE_AMSEL_R  &= ~PE4;       // AMSEL = 0

    // Configure PE5 (YP) for ADC use
    GPIO_PORTE_AFSEL_R  |=  PE5;       // AFSEL = 1
    GPIO_PORTE_DEN_R    &= ~PE5;       // DEN = 0
    GPIO_PORTE_DIR_R    &= ~PE5;       // DIR = 0
    GPIO_PORTE_AMSEL_R  |=  PE5;       // AMSEL = 1
```

```
    // Configure ADC to read from AIN9 (PE5, YP)
    ADC_SetChannel(8);

    // Take some samples to average
    for (i = 0; i < NUM_SAMPLES; i++){
        sum += ADC_Read();
    }

    // Compute average
    result = sum / NUM_SAMPLES;

    return result;
}

//coord Touch_GetCoords(void){
//    coord result;
//    short sumX = 0;
//    short sumY = 0;
//    short i = 0;

//    xVals[filterCounter] = Touch_ReadXVal();
//    yVals[filterCounter] = Touch_ReadYVal();
//
//    if (numVals < NUM_VALS_TO_AVG)
//        numVals++;
//
//    filterCounter++;
//    if (filterCounter >= NUM_VALS_TO_AVG)
//        filterCounter = 0;
//
//    for (i = 0; i < numVals; i++){
//        sumX += xVals[i];
//        sumY += yVals[i];
//    }
//
//    result.x = sumX / numVals;
//    result.y = sumY / numVals;
//
//    return result;
//}

void EnableInterrupts(void){
  GPIO_PORTA_IM_R |= (PA3 | PE5);
}

void DisableInterrupts(void){
  GPIO_PORTA_IM_R &= ~(PA3 | PE5);
}

void Touch_BeginWaitForTouch(void){
```

```
    // XP = 1  XN = DIG IN, INT ON FALL EDGE YP = Hi-Z YN = 0
    DisableInterrupts();

    // Set XP high
    TOUCH_XP = 0xFF;

    // Set YN low
    TOUCH_YN = 0x00;

    // Configure XN (PA3) for digital input
    GPIO_PORTA_DIR_R &= ~0x08;

    // Configure YP (PE5) for analog Hi-Z
    GPIO_PORTE_DIR_R &= ~0x20;
    GPIO_PORTE_DEN_R &= ~0x20;

    // Setup falling edge interrupt on XN (PA3)
    GPIO_PORTA_PUR_R |=  0x08;   // enable weak pull up
    GPIO_PORTA_IS_R  &= ~0x08;     // (d) PF4 is edge-sensitive
    GPIO_PORTA_IBE_R &= ~0x08;    //     PF4 is not both edges
    GPIO_PORTA_IEV_R &= ~0x08;    //     PF4 falling edge event
    GPIO_PORTA_ICR_R  =  0x08;      // (e) clear flag4
    GPIO_PORTA_IM_R  |=  0x08;      // (f) arm interrupt on PF4


    NVIC_PRI0_R = (NVIC_PRI7_R&0xFFFFFF00)|0x000000a0;
    NVIC_EN0_R = 1;  // (h) enable IRQ=0, interrupt 16 in NVIC


    EnableInterrupts();
}

long Touch_GetCoords(void){
    long result, temp, xPos, yPos;

    long cal[7] = {
      280448,//   86784,          // M0
        -3200,// -1536,          // M1
        -220093760,//-17357952,      // M2
        -3096,//-144,           // M3
        -275592,//-78576,         // M4
        866602824,// 69995856,       // M5
        2287498 //201804,         // M6
    };

    xPos = Touch_XVal;
    yPos = Touch_YVal;

    temp = (((xPos * cal[0]) + (yPos * cal[1]) + cal[2]) / cal[6]);
    yPos =(((xPos * cal[3]) + (yPos * cal[4]) + cal[5]) / cal[6]);
    xPos = temp;
```

```
    result = xPos << 16;
    result |= yPos;

    return result;
}


bool in_switch1(short xPos, short yPos){
    return DETECT_SWITCH1_XPOS_LEFT > xPos &&
           DETECT_SWITCH1_XPOS_RIGHT < xPos &&
           DETECT_SWITCH1_YPOS_UP < yPos &&
           DETECT_SWITCH1_YPOS_DOWN > yPos;
}

bool in_switch2(short xPos, short yPos){
  bool temp1 = DETECT_SWITCH2_XPOS_LEFT > xPos;
  bool temp2 = DETECT_SWITCH2_XPOS_RIGHT < xPos;
  bool temp3 = DETECT_SWITCH2_YPOS_UP < yPos;
  bool temp4 = DETECT_SWITCH2_YPOS_DOWN > yPos;
//  printf("%d %d %d %d", temp1, temp2, temp3, temp4);
    return DETECT_SWITCH2_XPOS_LEFT > xPos &&
           DETECT_SWITCH2_XPOS_RIGHT < xPos &&
           DETECT_SWITCH2_YPOS_UP < yPos &&
           DETECT_SWITCH2_YPOS_DOWN > yPos;
}

// unused at the moment, previously used to implement touch sensing on edge (not
void GPIO_PortA_Handler(void){
  GPIO_PORTA_ICR_R |= 0xFF;      // acknowledge flag4
////  Touched = 1;
//  if (Touch_ReadZ2() < 4000){
////    unsigned long tempZ = Touch_ReadZ2();
////    printf("%d\n", tempZ);
//    unsigned long tempX = Touch_ReadX();
//    unsigned long tempY = Touch_ReadY();
//
////    printf("raw: %d %d\n", tempX, tempY);
//    long temp = Touch_GetCoords();
//    short yPos = temp & 0xFFFF;
//    short xPos = (temp >> 16) & 0xFFFF;
//    printf("%d %d\n", xPos, yPos);
//
////    if (in_switch1(xPos, yPos) || in_switch2(xPos, yPos)){
////        TIMER0_CTL |= 0x01;
////    }
//  }
}
```

4.2 Task 1

```
/*
   task 1
   this will turn the lcd in to blue
   if want to change to another color
    check out the color table
*/


#include "SSD2119.h"
#include "tm4c123gh6pm.h"

// prototype
void LCD_Init(void);
void LCD_ColorFill(unsigned short color);

// 4 bit Color   red,green,blue to 16 bit color
// bits 15-11 5 bit red
// bits 10-5  6-bit green
// bits  4-0  5-bit blue
// color table
/*
unsigned short const Color4[16] = {
  0,                                                 //0 ?black                    (#0
 ((0x00>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3),  //1 ?blue                     (#0
 ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0x00>>3),  //2 ?green                    (#0
 ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0xAA>>3),  //3 ?cyan                     (#0
 ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0x00>>3),  //4 ?red                      (#A
 ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3),  //5 ?magenta                  (#A
 ((0xAA>>3)<<11) | ((0x55>>2)<<5) | (0x00>>3),  //6 ?brown                    (#A
 ((0xAA>>3)<<11) | ((0xAA>>2)<<5) | (0xAA>>3),  //7 ?white / light gray       (#A
 ((0x55>>3)<<11) | ((0x55>>2)<<5) | (0x55>>3),  //8 ?dark gray /bright black (#5
 ((0x55>>3)<<11) | ((0x55>>2)<<5) | (0xFF>>3),  //9 ?bright blue              (#5
 ((0x55>>3)<<11) | ((0xFF>>2)<<5) | (0x55>>3),  //10 ?bright green            (#5
 ((0x55>>3)<<11) | ((0xFF>>2)<<5) | (0xFF>>3),  //11 ?bright cyan             (#5
 ((0xFF>>3)<<11) | ((0x55>>2)<<5) | (0x55>>3),  //12 ?bright red              (#F
 ((0xFF>>3)<<11) | ((0x55>>2)<<5) | (0xFF>>3),  //13 ?bright magenta          (#F
 ((0xFF>>3)<<11) | ((0xFF>>2)<<5) | (0x55>>3),  //14 ?bright yellow           (#F
 ((0xFF>>3)<<11) | ((0xFF>>2)<<5) | (0xFF>>3)   //15 ?bright white            (#F
};
*/

int main()
{
  LCD_Init();
  while(1){
    LCD_ColorFill(((0x00>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3));
  }
  return 0;
}
```

4.3 Task 2

```
// task 2
#include "my_header.h"
#include "SSD2119.h"
#include "tm4c123gh6pm.h"

// prototype
void adc_init(void);
void ADC0_Handler(void);
void timer_init(void);

float temp;
int main()
{
  LCD_Init();
  timer_init();
  adc_init();
  while(1){
  }
  return 0;
}

void timer_init(void){
  // enable timer 0
  RCGCTIMER = 0x01;
  // disable the timer. bit 0 GPTM Timer A enable
  TIMER0_CTL &= 0x0;
  // select 32-bit mode
  // 0x0 For a 16/32-bit timer, this value selects the 32-bit timer configuratio
  TIMTER0_CONFIG = 0x0;
  // config TAMR bit to be in periodic timer mode
  // value 0x2 periodic timer mode
  TIMER0_TAMR = 0x2;
  // TACDIR bit to count down. 0 for count down
  TIMER0_TAMR &= ~0x10;
  // value 16,000,000
  TIMER0_TAILR = 0xF42400;
  // enable the timer
  TIMER0_CTL |= 0x21;
}

void adc_init(void){
  volatile unsigned long delay;
  // enable module 0
  RCGCADC |= 0x1;
  // a delay before sampling
  delay = RCGCADC;
  // disable sample sequencer 3 ADCACTSS
  ADC0_SEQ &= ~ASEN3;
  // select the trigger for timer
  ADC0_MUX &= ~EMUX_EM3;
```

```
  ADC0_MUX |= EMUX_TIMER;
  // select adc input channel ADCSSMUX3
  ADC0_SS_MUX3 &= 0x0;
  // one ended, raw interrupt enable, temp sensor read
  ADC0_SS_CTL3 |= 0xE;

        // clear the interrupt status for potential leftover
  ADC0_ISC |= (1 << 3);
  // interrupt #17
  NVIC_EN0 |= (1 << 17);
  // If interrupts are to be used,
  // set the corresponding MASK bit in the ADCIM register
  ADC0_IM |= (1 << 3);
  // enable SS3
  ADC0_SEQ |= ASEN3;
}

void ADC0_Handler(void){
    // start new conversion ADCPSSI
  ADC0_SS_INIT |= (1 << 3);
  while (ADC0_RIS & (1 << 3) == 0){}
  temp = 147.5 - ((247.5 * ADC0_FIFO3) / 4096.0);
//   printf("%2.2f\n", temp);
  // clear the interrupt
  ADC0_ISC |= (1 << 3);
  LCD_PrintFloat(temp);
}
```

4.4 Task 3

```
// task 3
#include "my_header.h"
#include "tm4c123gh6pm.h"
#include "SSD2119.h"
#include "stdbool.h"

// prototype
typedef enum {OFF, STOP, WARN, GO}state;
void LCD_DrawCircle(unsigned short x0, unsigned short y0, unsigned short radius,
void LCD_DrawFilledCircle(unsigned short x0, unsigned short y0, unsigned short r
void LCD_DrawPixel(unsigned short x, unsigned short y, unsigned short color);
void LCD_DrawRect(unsigned short x, unsigned short y, short width, short height,
void ui_define(void);
void Touch_BeginWaitForTouch(void);
void timer0_init(void);
void timer1_init(void);
void interrupt_init(void);
void set_state(state);

/*
unsigned short const Color4[16] = {
  0,                                              //0 Â¡V black
```

```
 ((0x00>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3),  //1 Â¡V blue
 ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0x00>>3),  //2 Â¡V green
 ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0xAA>>3),  //3 Â¡V cyan
 ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0x00>>3),  //4 Â¡V red
 ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3),  //5 Â¡V magenta
 ((0xAA>>3)<<11) | ((0x55>>2)<<5) | (0x00>>3),  //6 Â¡V brown
 ((0xAA>>3)<<11) | ((0xAA>>2)<<5) | (0xAA>>3),  //7 Â¡V white / light gray
 ((0x55>>3)<<11) | ((0x55>>2)<<5) | (0x55>>3),  //8 Â¡V dark gray /bright black
 ((0x55>>3)<<11) | ((0x55>>2)<<5) | (0xFF>>3),  //9 Â¡V bright blue
 ((0x55>>3)<<11) | ((0xFF>>2)<<5) | (0x55>>3),  //10 Â¡V bright green
 ((0x55>>3)<<11) | ((0xFF>>2)<<5) | (0xFF>>3),  //11 Â¡V bright cyan
 ((0xFF>>3)<<11) | ((0x55>>2)<<5) | (0x55>>3),  //12 Â¡V bright red
 ((0xFF>>3)<<11) | ((0x55>>2)<<5) | (0xFF>>3),  //13 Â¡V bright magenta
 ((0xFF>>3)<<11) | ((0xFF>>2)<<5) | (0x55>>3),  //14 Â¡V bright yellow
 ((0xFF>>3)<<11) | ((0xFF>>2)<<5) | (0xFF>>3)   //15 Â¡V bright white
};
*/
#define LCDBLACK 0x0
#define LCDBLUE ((0x00>>3)<<11) | ((0x00>>2)<<5) | (0xAA>>3)
#define LCDGREEN ((0x00>>3)<<11) | ((0xAA>>2)<<5) | (0x00>>3)
#define LCDRED ((0xAA>>3)<<11) | ((0x00>>2)<<5) | (0x00>>3)
state system;
// LCD_drawrect and draw filled rect draw linework
bool pressed;
int main()
{
  pressed = false;
  LCD_Init();
  ui_define();
  system = OFF;
  set_state(system);

  Touch_Init(); // include ADC_Init();

  interrupt_init();
  timer0_init();
  timer1_init();
  Touch_BeginWaitForTouch();
  while(1){
//    unsigned long tempZ = Touch_ReadZ2();
//    printf("%d\n", tempZ);
    if (Touch_ReadZ2() < 3400){

      unsigned long tempX = Touch_ReadX();
      unsigned long tempY = Touch_ReadY();

//      printf("raw: %d %d\n", tempX, tempY);
      long temp = Touch_GetCoords();
      short yPos = temp & 0xFFFF;
      short xPos = (temp >> 16) & 0xFFFF;
      LCD_SetCursor(0, 0);
```

```
      printf("%d %d\n", xPos, yPos);
//      if (in_switch2(xPos, yPos)) {
//        printf("true");
//      }

      if (in_switch1(xPos, yPos) || in_switch2(xPos, yPos)){
            TIMER0_IMR |= 0x1;
            TIMER0_ICR |= 0x1;
            TIMER0_CTL |= 0x1;
      }
    }
  }
  return 0;
}

void interrupt_init(void){
  NVIC_EN0 |= (1 << 21) | (1 << 19);
}

 void timer1_init(void){
   // enable timer 0
   RCGCTIMER |= 37;
   // disable the timer. bit 0 GPTM Timer A enable
   TIMER1_CTL &= ~0x00000001;
   // select 32-bit mode
   // 0x0 For a 16/32-bit timer, this value selects the 32-bit timer configurati
   TIMTER1_CONFIG = 0x0;
   // config TAMR bit to be in periodic timer mode
   // value 0x2 periodic timer mode
   TIMER1_TAMR = 0x2;
   // TACDIR bit to count down. 0 for count down
   TIMER1_TAMR |= 0x10;
   TIMER1_IMR |= 0x1;
   TIMER1_TAILR = 0x4C4B400;
   // this is where I stuck
   // need to clear out the value for potential previous
   // left value
   TIMER1_ICR = 0x01;
 }

 void timer0_init(void){
   // enable timer 0
   RCGCTIMER |= 35;
   // disable the timer. bit 0 GPTM Timer A enable
   TIMER0_CTL &= ~0x00000001;
   // select 32-bit mode
   // 0x0 For a 16/32-bit timer, this value selects the 32-bit timer configurati
   TIMTER0_CONFIG = 0x0;
   // config TAMR bit to be in periodic timer mode
   // value 0x2 periodic timer mode
   TIMER0_TAMR = 0x2;
```

```
   // TACDIR bit to count down. 0 for count down
   TIMER0_TAMR |= 0x10;

   TIMER0_IMR |= 0x1;
   TIMER0_TAILR = 0x1E84800;
   // this is where I stuck
   // need to clear out the value for potential previous
   // left value
   TIMER0_ICR = 0x01;
 }

void set_state(state color){
  switch(color){
  case OFF:
    LCD_DrawFilledCircle(RED_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(YELLOW_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(GREEN_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    break;
  case STOP:
    LCD_DrawFilledCircle(YELLOW_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(GREEN_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(RED_XPOS, BULB_YPOS, RADIUS - 1, LCDRED);
    break;
  case WARN:
    LCD_DrawFilledCircle(RED_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(GREEN_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(YELLOW_XPOS, BULB_YPOS, RADIUS - 1, LCDGREEN | LCDRED);
    break;
  case GO:
    LCD_DrawFilledCircle(RED_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(YELLOW_XPOS, BULB_YPOS, RADIUS - 1, LCDBLACK);
    LCD_DrawFilledCircle(GREEN_XPOS, BULB_YPOS, RADIUS - 1, LCDGREEN);
    break;
  }
}


// check 2 sec
void Timer0_Handler(void){
  TIMER0_ICR_R |= 0x1; // clear timeout flag
  Touch_ReadX();
  Touch_ReadY();
  long temp = Touch_GetCoords();
  short yPos = temp & 0xFFFF;
  short xPos = (temp >> 16) & 0xFFFF;
  LCD_SetCursor(0, 0);
  printf("%d %d\n", xPos, yPos);
  if (in_switch1(xPos, yPos)){
    if (system == OFF){
      TIMER1_ICR |= 0x01;
      TIMER1_CTL |= 0x01;
```

```
      system = STOP;
    }else{
      TIMER1_ICR |= 0x01;
      TIMER1_CTL &= ~0x01;
      system = OFF;
    }
  }
  else if(in_switch2(xPos, yPos)){
    if (system == GO){
      TIMER1_ICR |= 0x01;
      system = WARN;
    }
  }
  TIMER0_IMR &= ~0x1;
  TIMER0_ICR |= 0x01;
  TIMER0_CTL &= ~0x01;
  set_state(system);
}

// check 5 sec
void Timer1_Handler(void){
  TIMER1_ICR_R |= TIMER_ICR_TATOCINT; // clear timeout flag
  switch(system){
  case STOP:
    system = GO;
    break;
  case WARN:
    system = STOP;
    break;
  case GO:
    system = STOP;
    break;
  }
  set_state(system);
}

void ui_define(void){
  // pre layout three bulbs
  LCD_DrawCircle(RED_XPOS, BULB_YPOS, RADIUS, LCDRED);
  LCD_DrawCircle(YELLOW_XPOS, BULB_YPOS, RADIUS, LCDRED | LCDGREEN);
  LCD_DrawCircle(GREEN_XPOS, BULB_YPOS, RADIUS, LCDGREEN);

  // User Switches
  LCD_DrawFilledRect(SWITCH1_XPOS, SWITCH_YPOS, SWITCH_WIDTH, SWITCH_HEIGHT, LCD
  LCD_DrawFilledRect(SWITCH2_XPOS, SWITCH_YPOS, SWITCH_WIDTH, SWITCH_HEIGHT, LCD

  // print the string
  LCD_SetCursor(65, 165);
  LCD_PrintString("Start/Stop");
  LCD_SetCursor(195, 165);
  LCD_PrintString("Pedestrian");
```

```
}
```